Chapitre 1 : Les bases de Python

1. Prise en main de Python

Pour **ouvrir Python :** choisir **Spyder**

Deux possibilités pour utiliser Python :
_ la console : elle affiche les résultats des différents programmes, mais peut aussi être utilisée comme "super-calculatrice" ou pour les instructions simples
_ l'éditeur de texte : permet d'écrire des programmes plus élaborés, de les enregistrer, puis de les exécuter
Dans cet éditeur, on peut ajouter des commentaires, à l'aide de **#**

Python contient dans son noyau un certain nombre d'instructions et de fonctions.
Pour une utilisation plus mathématique, il est souvent utile d'importer des "modules" ou "bibliothèques" supplémentaires.
Nous utiliserons très souvent les modules numpy, mathplotlib.pyplot, numpy.random, pandas, numpy.linalg.

Pour importer un module :
import nomdumodule
Dans ce cas, pour utiliser une fonction de ce module : **nomdumodule.nomfonction**
Ou avec un alias :
import nomdumodule as alias
Dans ce cas, pour utiliser une fonction de ce module : **alias.nomfonction**

Ex : import numpy as np
Ensuite, on peut utiliser : np.exp, …

Autre possibilité, importer uniquement une fonction : **from numpy import exp**
ou toutes les fonctions : **from numpy import ***
Dans ce cas, le préfixe est inutile, on peut utiliser simplement exp

Affichage : Dans la console : il suffit d'écrire le nom de la variable ou le calcul à effectuer
Dans l'éditeur de texte : utiliser **print** pour afficher une valeur (voir plus loin).

Python et maths :
Si besoin : **import numpy as np**

Opérations usuelles sur les réels	**+ , -, *, /**
Puissance (x^n)	**x ** n**
Quotient de a par b (division euclidienne)	**a // b**
Reste de la division euclidienne de a par b	**a % b**
Valeur absolue	**abs**
Partie entière	**int**
exp, ln, racine carrée	**np.exp np.log np.sqrt**
e, π	**np.e np.pi**

Dans le reste du chapitre, on supposera avoir introduit : import numpy as np

2. Les variables

2.1 Notion de variable

Une variable est définie par un nom (qui commence par une lettre, mais peut contenir des chiffres).
Elle peut contenir un nombre entier, un nombre réel, une chaine de caractère, une liste, une matrice…

Pour affecter une valeur à une variable, on utilise : **variable = valeur**

Attention, c'est la valeur de droite qui est affecté à la variable de gauche !
Ex : **x = y** affecte la valeur de y à x.

Pour demander une valeur à l'utilisateur, on utilise **input('question')**, dont le résultat est par défaut une chaine de caractère.
Pour demander un entier n à l'utilisateur, on écrira :

```
n=int(input('Valeur de n ? '))
```

Dans la console, la question "Valeur de ?" apparait, avec le curseur. L'utilisateur rentre une valeur, suivie de Entrée. Cette valeur est alors stockée dans la variable n.

Pour demander un réel x à l'utilisateur, on écrira :

```
x=float(input('valeur de x ? '))
```

Pour afficher la valeur d'une variable à l'écran :

print(x) ou **print('x vaut ',x)**

2.2 Listes

Pour créer une liste, 3 possibilités :
_ donner la liste des valeurs : **L = [x1, …, xn]**
Exemple : `L = [-1, 1, 2, 5]`

_ donner les bornes et le pas :
Si a, b et h sont des entiers :
L = list(range(a, b, h)) crée la liste [a;a + h;a + 2h;….] qui s'arrête au dernier nombre du type a + nh qui est < b. (h est le pas de la subdivision)
L = list(range(a, b)) créé la liste [a, a+1, …, b – 1]
L = list(range(b)) créé la liste [0, …, b – 1]
Attention, dans chaque cas, b n'est pas la liste !

_ A l'aide d'une expression : **L=[expression for … in …]**

Ex : Liste des nombres impairs de 1 à 99 de deux manières différentes :

Rappels sur les listes :
Soit L une liste.
_ Si i un entier, **L[i]** est le i-ème coefficient de L (attention, l'indice du premier coefficient est 0 !).
_ tous les éléments de L : **L[:]**
_ longueur d'une liste L : **len(L)**
_ Liste vide : **L=[]**
_ Ajouter x à la fin de L : **L.append(x)**
_ Supprimer L(k) : **del L[k]** (attention, cette opération décale les indices suivants !)
_ Appartenance d'un élément à une liste : **x in L** ⟶ True ou False
_ **L.count(x)** : nombre d'occurrences de x dans L

3. Instructions conditionnelles

3.1 Booléens et tests

Un booléen est un nombre qui peut valoir : vrai (`True`) ou faux (`False`). Il est souvent obtenu à l'aide d'un test.

Si x et y sont des nombres :

x==y : teste si x et y sont égaux
x!=y : teste si x et y sont différents
x<y et **x>y** : évidents
x>=y teste si $x \geq y$
x<=y teste si $x \leq y$

Si a et b sont des booléens :
not(a) ("non a") est vrai si et seulement si a est faux
a & b ou **a and b** ("a et b") est vrai si et seulement si a et b sont vrais
a | b ou **a or b** ("a ou b") est vrai si et seulement si au moins un des deux est vrai.

3.2 L'instruction IF

Syntaxes :

```
if condition :
    instruction          # Si … alors …

if condition :
    instruction1
else :
    instruction2         # Si … alors … sinon …

if condition1 :
    instruction1
elif condition2 :
    instruction2
else :
    instruction3         # Si … alors … sinon si … sinon …
```

instruction1 est effectuée si condition1 est vraie, instruction2 est effectuée si condition1 est fausse et condition2 vraie, instruction3 est effectuée si condition1 et condition2 sont fausses.

Exemple : Soit x un réel. Tester si $x \in [0;1]$ (on affichera 'oui' ou 'non').

4. Les boucles

1) Pour définir un compteur :

Initialisation (avant la boucle) : **i =0**
Relation de récurrence (dans la boucle) : **i = i + 1** ou **i +=1**

2) Pour effectuer plusieurs fois la même instruction, on effectue une boucle.

_ soit **on connaît par avance le nombre d'itérations** : on utilise **FOR**
_ soit **on ne connaît pas par avance le nombre d'itérations :** on utilise **WHILE**

Syntaxe :

```
for i in range(…) :
        instruction

while condition :
        instruction
```

Remarques :
_ pour les différentes syntaxes de range, voir le paragraphe 2.2
_ attention à l'indentation : les lignes indentées seront répétées. A partir de la ligne non indentée, on sort de la boucle.
_ avec while, il n'y a pas de compteur. Si vous avez besoin d'un compteur (pour compter le nombre d'itérations effectuées), c'est à vous de le rajouter.

Exemples :
1) Afficher les carrés des 50 premiers entiers naturels non nuls

2) Afficher les cubes des entiers naturels qui sont inférieurs à 10000

Remarque : De manière générale, si L est une liste, on peut utiliser : **for x in L**
Dans ce cas, x prendra successivement toutes les valeurs de L.

5. Sommes et produits

5.1 Calcul de somme :

Si on veut calculer $S = \sum u_k$:

Méthode 1 : Avec une boucle

Initialisation : **S =0**
Formule de récurrence : **S = S + uk** (ou : S+=uk)

Exemple : Calcul de $S = \sum_{i=1}^{1000} \frac{1}{i}$

Méthode 2 : Avec np.sum

Si L est une liste, **np.sum(L)** retourne la somme de tous les coefficients de L.

Exemple précédent :

Remarque : Si $\forall\, n \in \mathbb{N}$, $S_n = \sum_{k=0}^{n} u_k$, alors $S_0 = u_0$

Sommes cumulées avec **np.cumsum** :

Si $x = [x_1, x_2, \dots, x_n]$, alors **np.cumsum(x) =** $[x_1, x_1 + x_2, x_1 + x_2 + x_3, \dots, x_1 + x_2 + \dots + x_n]$.

Ex : Si x = [1, 5, 3, 8]

Remarque : si u est un vecteur qui contient les premiers termes $(u_0, \dots, u_n)$ d'une suite, alors numpy.cumsum(u) contient les premiers termes de la **somme partielle** correspondante.

5.2 Calcul de produits

Si on veut calculer $P = \prod u_k$:

Initialisation : **P =1**
Formule de récurrence : **P = P * uk** (ou : P*=uk)

Ex : Pour $n \geq 1$, soit $P_n = \prod_{k=1}^{n} \left(1 + \frac{1}{\sqrt{k}}\right)$. Calculer le premier entier n tel que $P_n > 10^6$

Remarque :
Il existe une fonction **np.prod** qui s'utilise comme np.sum.